ERLANGEN REGIONAL
COMPUTING CENTER

If it doesn’t work, we learn something

Instructive case studies in Performance Engineering

Georg Hager
Erlangen Regional Computing Center (RRZE), Erlangen, Germany

SIAM PP18
March 7-10, 2018
Tokyo, Japan

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Motivation

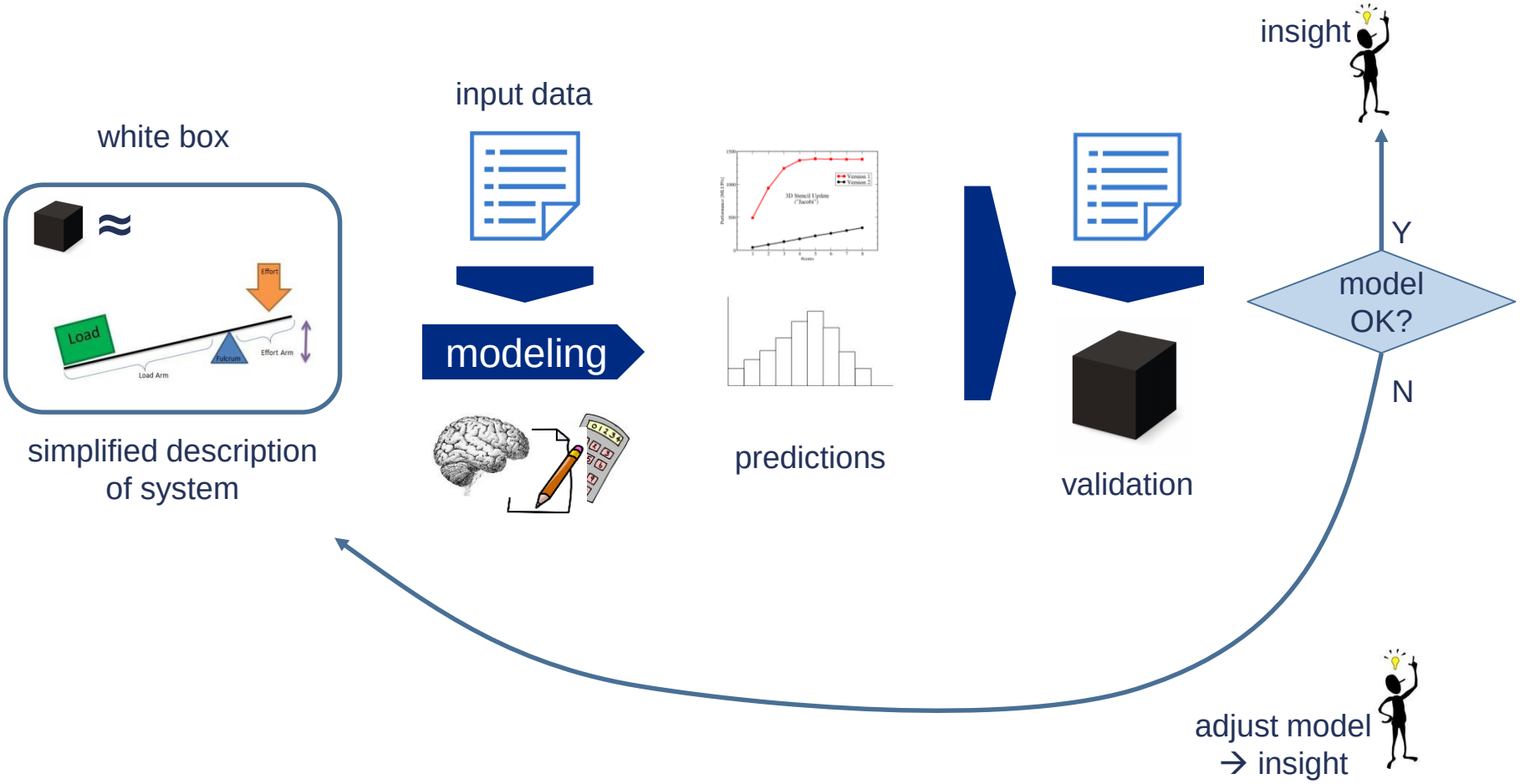
- Analytic performance modeling:

Constructing a simplified model for the interaction between software and hardware in order to understand lowest-order performance behavior

- Basic questions addressed by analytic performance models
  - What is the bottleneck?
  - What is the next bottleneck after optimization?
  - Impact of processor frequency and socket scalability → energy efficiency

- What if the model fails?
  - We learn something
  - We may still be able to use the model in a less predictive way

The Performance Engineering cycle



Getting a little more specific

What data/knowledge can a model be based on?

1. Only documented hardware properties + hypotheses
   - Purely analytic model

   white box

2. Hardware properties + (some) microbenchmark results + hypotheses
   - (Partly) phenomenological model

   gray box

3. Measured performance/speedup data + hypotheses
   - Curve-fitting analytic model

   black box

Examples for white-/gray-box models

$$S(N) = \frac{1}{s + \frac{1-s}{N} + c(N)}$$

Amdahl's Law with communication

$$T_{PtP} = T_l + \frac{L}{B}$$

Hockney model for message transmission time

$$T_{\text{exec}} = \max(T_{\text{calc}}, T_{\text{data}})$$

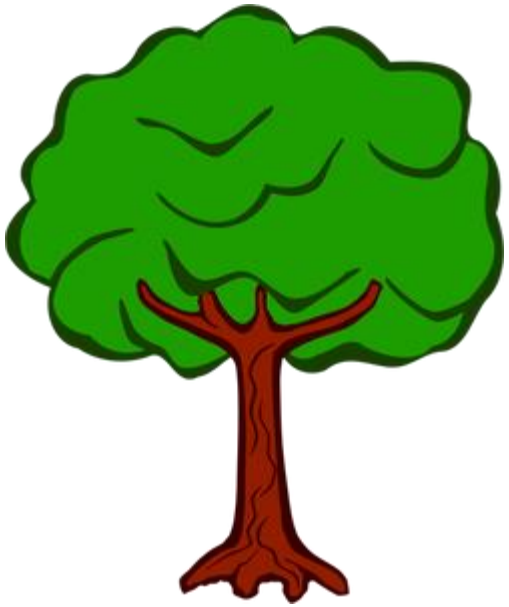
Roofline model for loop code execution time

$$T_{\text{exec}} = f(T_{nOL}, T_{\text{data}}, T_{OL})$$

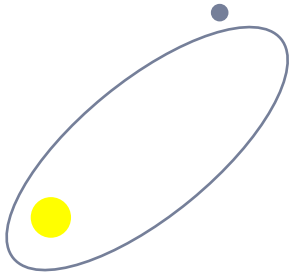
ECM model for loop code execution time

An example from physics

$$h(t) = h_0 - \frac{1}{2} g\, t^2$$



$$\vec{F}_{12} = \frac{\gamma m_1 m_2}{r_{12}^3} \vec{r}_{12}$$



“IF THE MODEL DOES NOT WORK, WE LEARN SOMETHING”

Three (and a half) examples

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Example: The Haswell port 7 AGU mystery

- Schönauer Vector Triad `A(:)=B(:)+C(:)*D(:)`
- Haswell Xeon E5 core

Port 0	Port 1	Port 2	Port 3	Port 4	Port 5	Port 6	Port 7
ALU	ALU	LOAD	LOAD	STORE	ALU	ALU	AGU
FMA	FMA	AGU	AGU		FSHUF	JUMP	
FMUL		32b ↑	32b ↑	32b ↓	JUMP		

- Expectation: 8 iterations in 3 cycles
  → $P_{\max} = 12.26$ GF/s @ 2.3 GHz

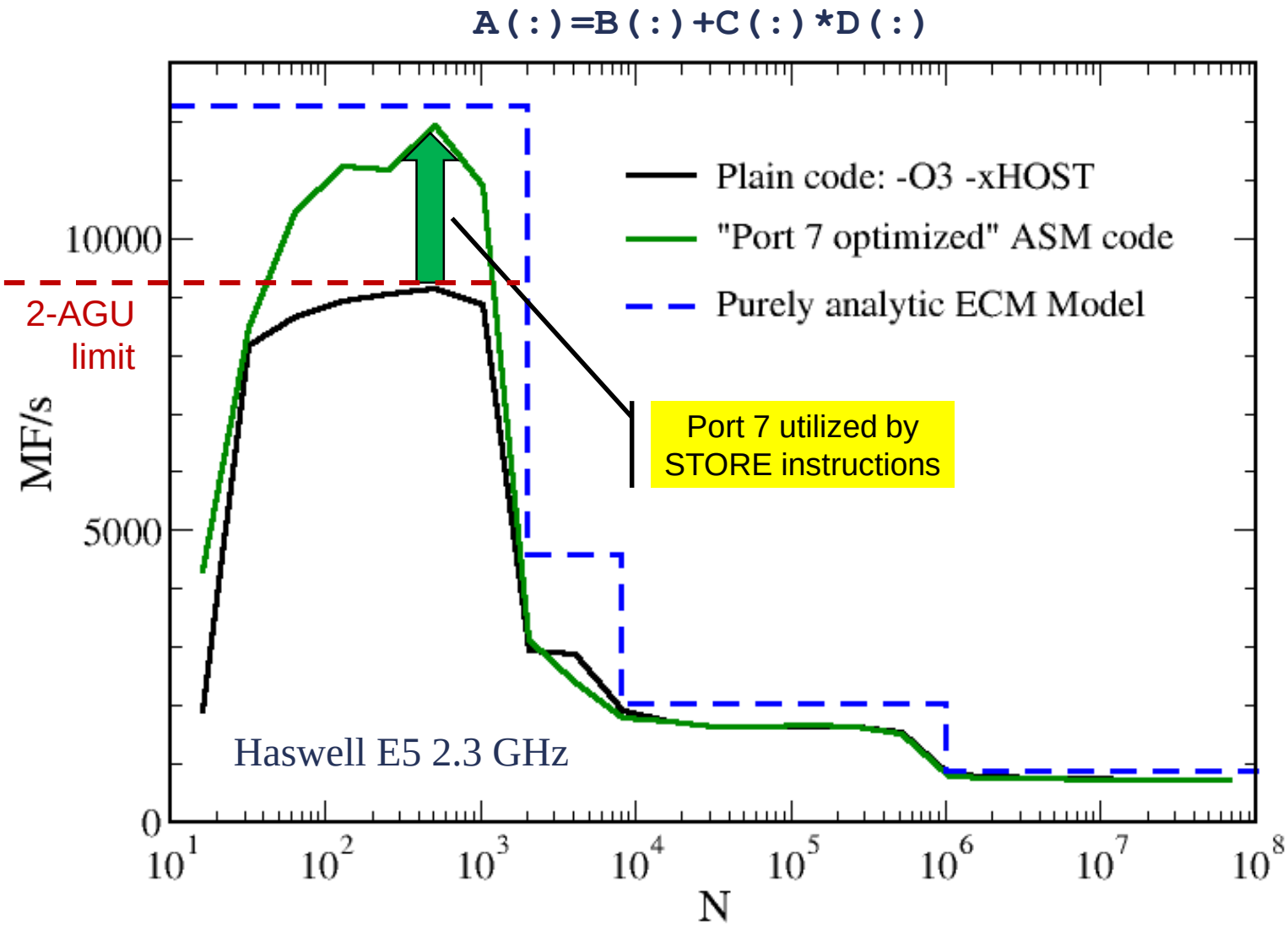
Generated assembly code (Intel V16up4)

- "Perfect code" at first sight
- Compiler only uses complex addressing mode
- Expected performance limit from port model:

  $P_{\max} = 12.3\ \mathrm{GFlop/s}$

```
vmovupd (%r12,%r10,8),%ymm2
vmovupd 0x20(%r12,%r10,8),%ymm4
vmovupd 0x40(%r12,%r10,8),%ymm6
vmovupd 0x60(%r12,%r10,8),%ymm8
vmovupd (%r15,%r10,8),%ymm1
vmovupd 0x20(%r15,%r10,8),%ymm3
vmovupd 0x40(%r15,%r10,8),%ymm5
vmovupd 0x60(%r15,%r10,8),%ymm7
vfmadd213pd (%rsi,%r10,8),%ymm1,%ymm2
vfmadd213pd 0x20(%rsi,%r10,8),%ymm3,%ymm4
vfmadd213pd 0x40(%rsi,%r10,8),%ymm5,%ymm6
vfmadd213pd 0x60(%rsi,%r10,8),%ymm7,%ymm8
vmovupd %ymm2,0x0(%r13,%r10,8)
vmovupd %ymm4,0x20(%r13,%r10,8)
vmovupd %ymm6,0x40(%r13,%r10,8)
vmovupd %ymm8,0x60(%r13,%r10,8)
add     $0x10,%r10
cmp     %r9,%r10
jb      4016a0 <do_triad_+0x360>
```

“If the model does not work, we learn something”



Port 7

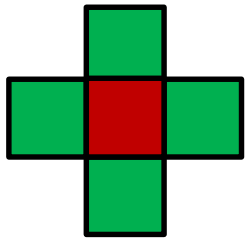
AGU

Simple addesses only!

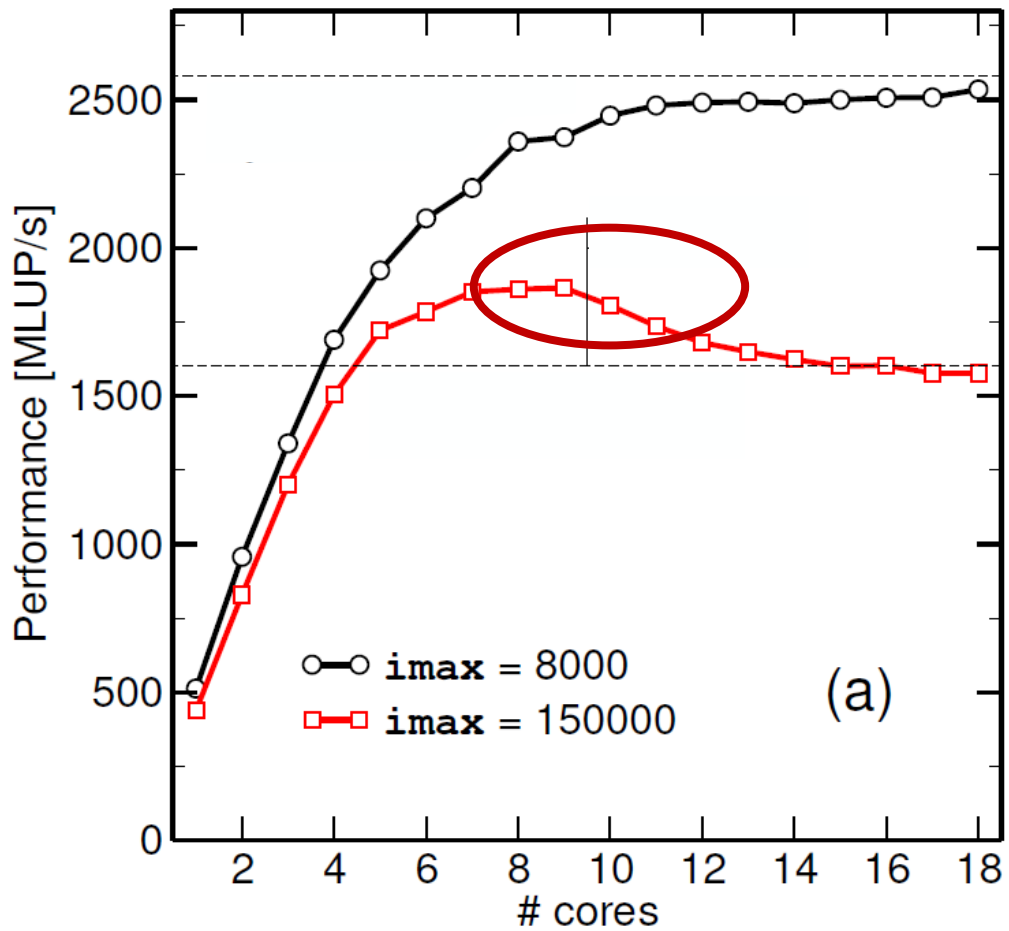
Example: Weird scaling behavior of a stencil code

```
double precision, dimension(1:imax,1:kmax,0:1) :: phi
integer :: t0,t1
t0 = 0 ; t1 = 1

!$OMP PARALLEL PRIVATE(it)
  do it = 1,itmax
!$OMP DO SCHEDULE(STATIC)
    do k = 2,kmax-1
      do i = 2,imax-1
         phi(i,k,t1) = (  phi(i+1,k,t0) + phi(i-1,k,t0)
                        + phi(i,k+1,t0) + phi(i,k-1,t0) ) * 0.25d0
      enddo
    enddo
!$OMP END DO
!$OMP SINGLE
    i = t0 ; t0 = t1 ; t1 = i
!$OMP END SINGLE
  enddo
!$OMP END PARALLEL
```



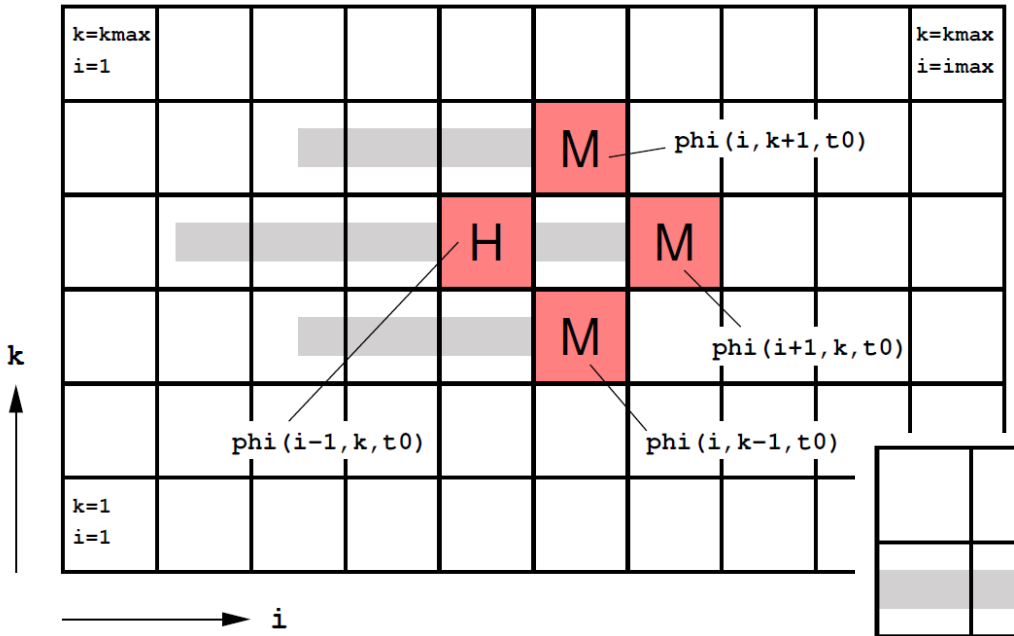
Scaling behavior changes with inner dimension



Intel Broadwell Xeon E5 non-CoD

Working set 1 GiB

Solution: layer conditions!

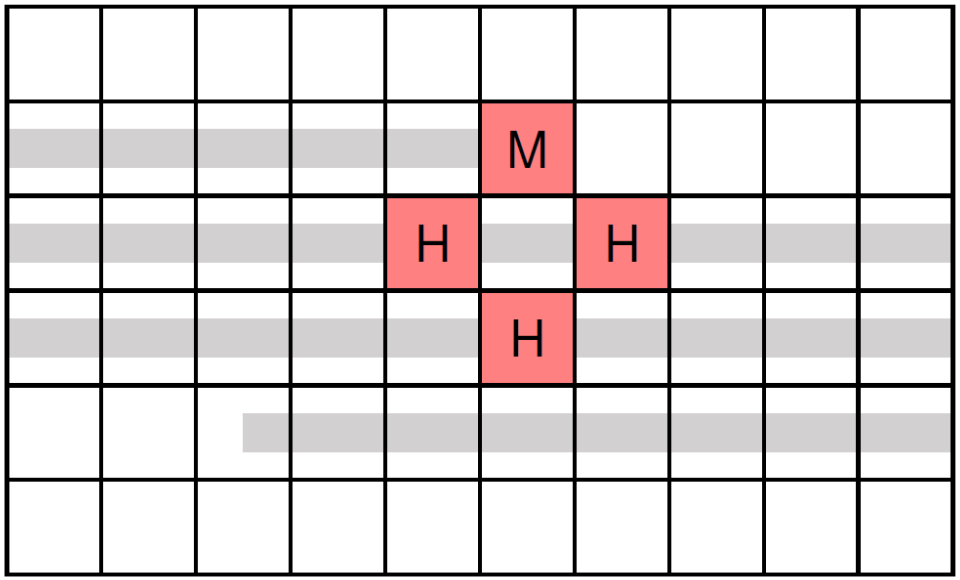


Cache k has size C_k

Layer condition (height r stencil):

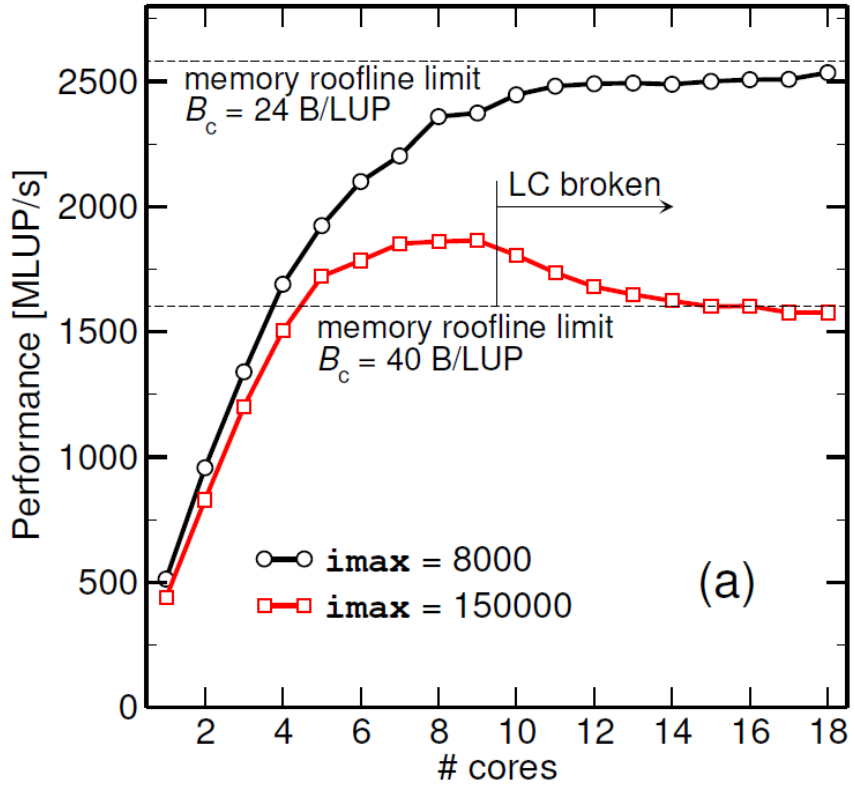
$$(2r+1) \cdot N_i \cdot 8\ \mathrm{B} < \frac{C_k}{2}$$

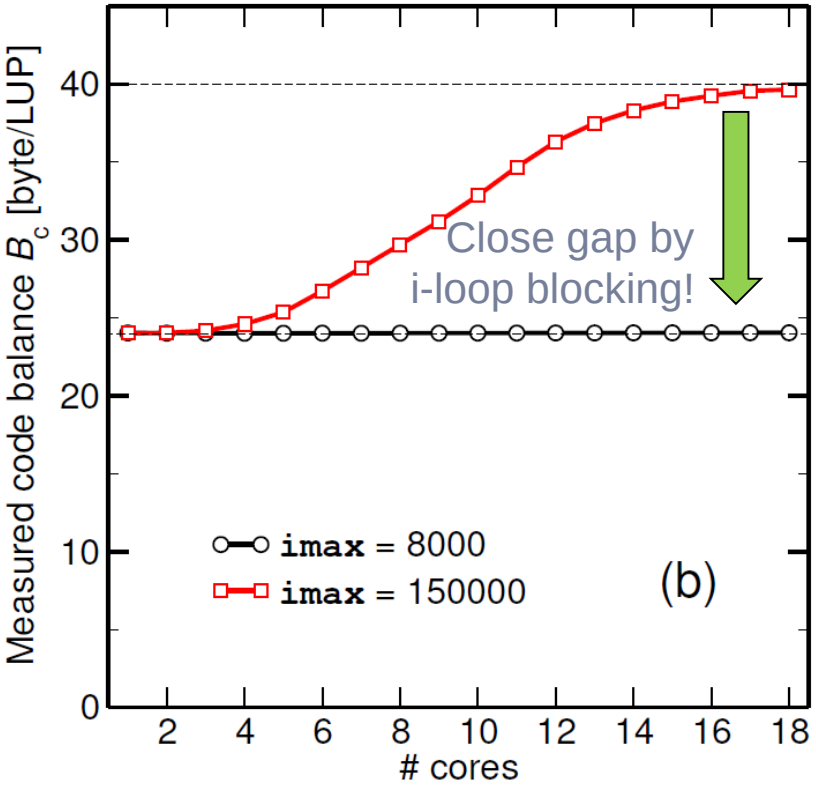
2D 5-pt: $r = 1$



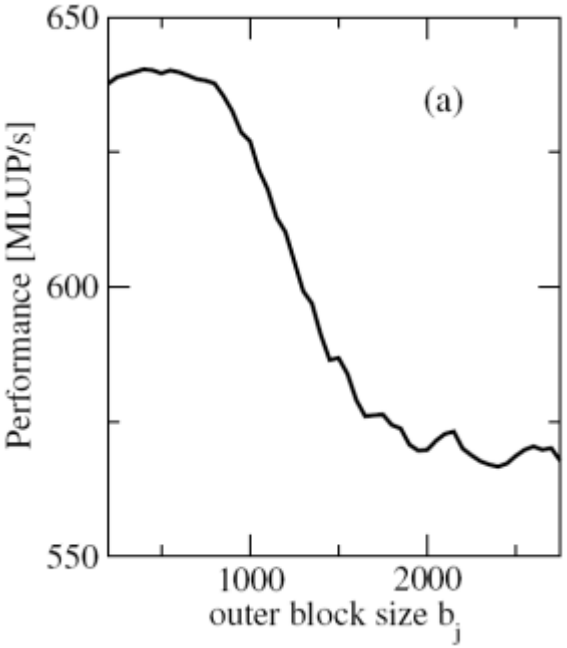
Validating the LC hypothesis

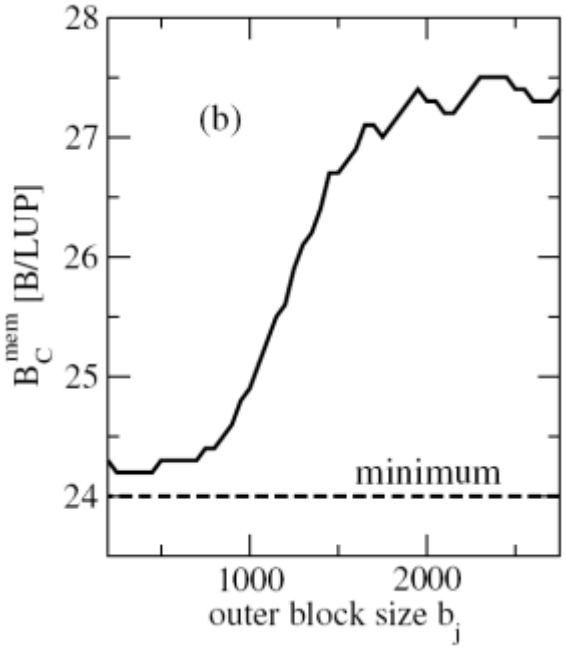
Measurement via likwid-perfctr (or PAPI)





2D 5-pt: Inner (i) loop blocking is not enough!

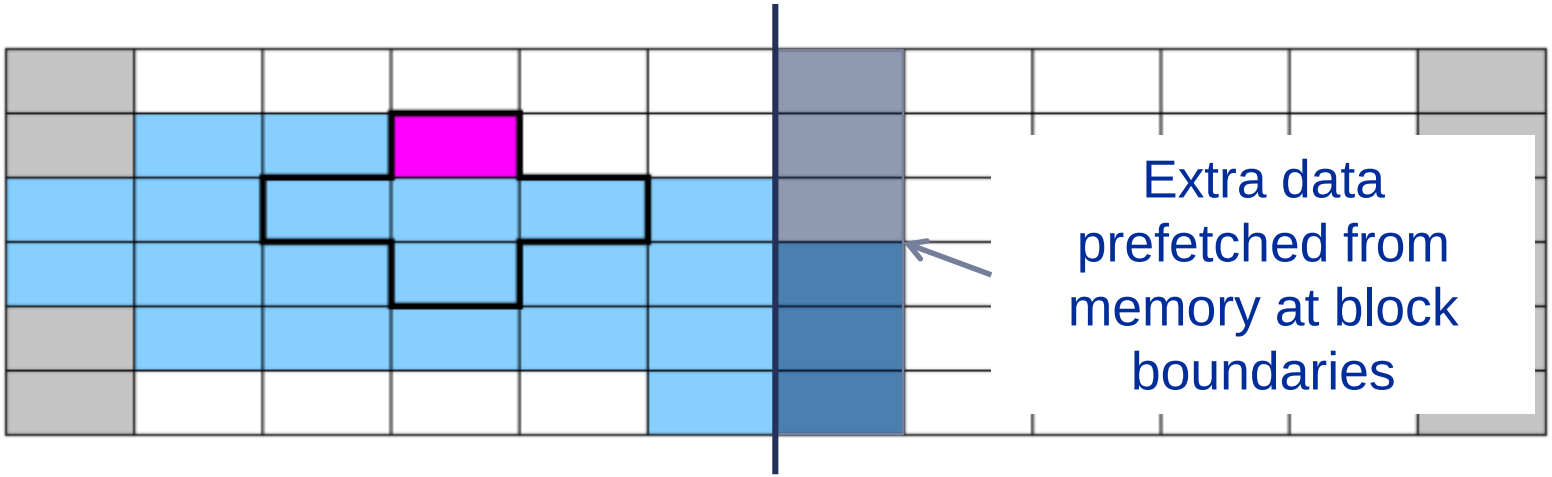




$$b_i = 800$$

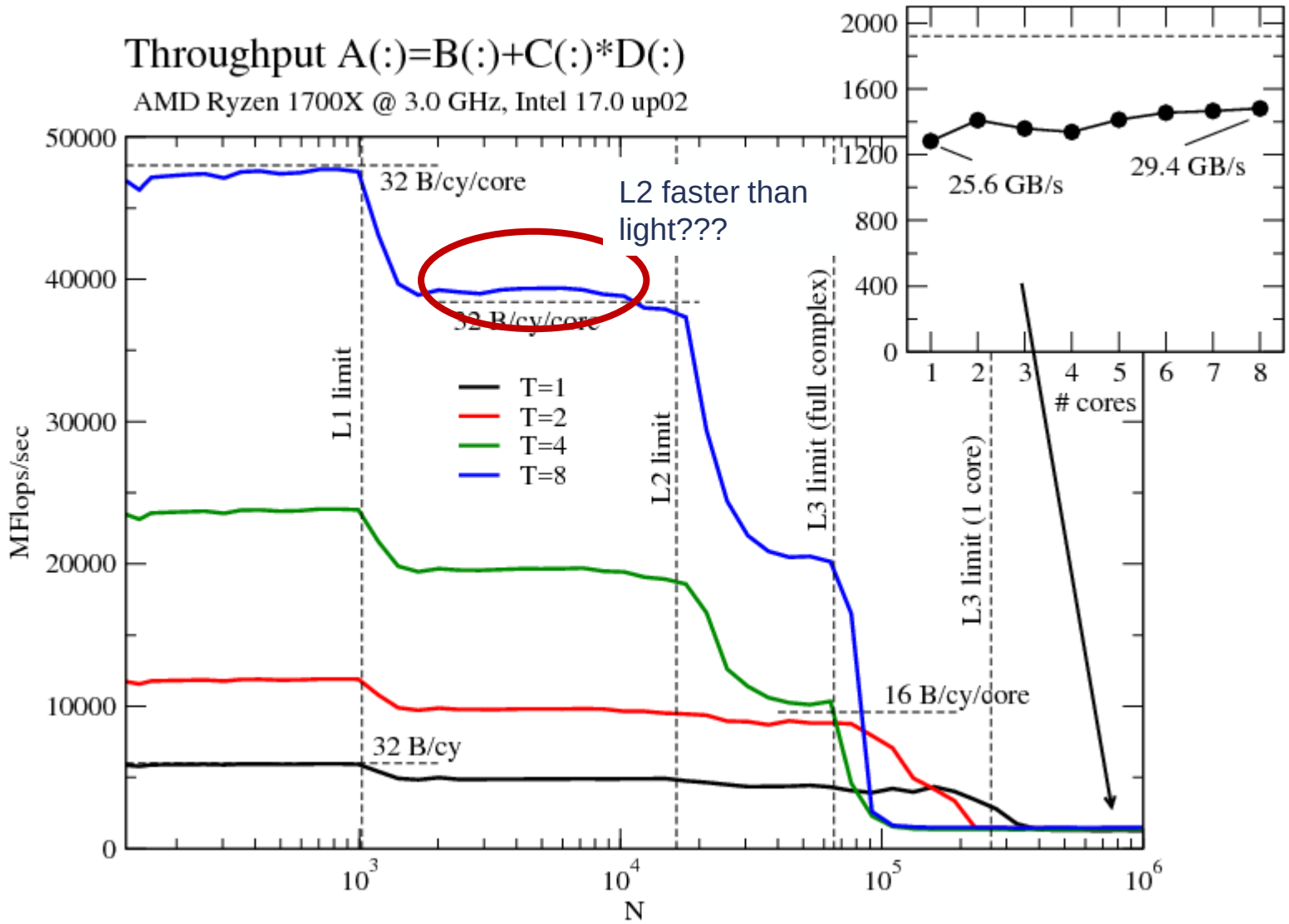
$$N_i = 3.5 \times 10^4$$

$$N_j = 1.2 \times 10^4$$



Another riddle

https://blogs.fau.de/hager/archives/7810

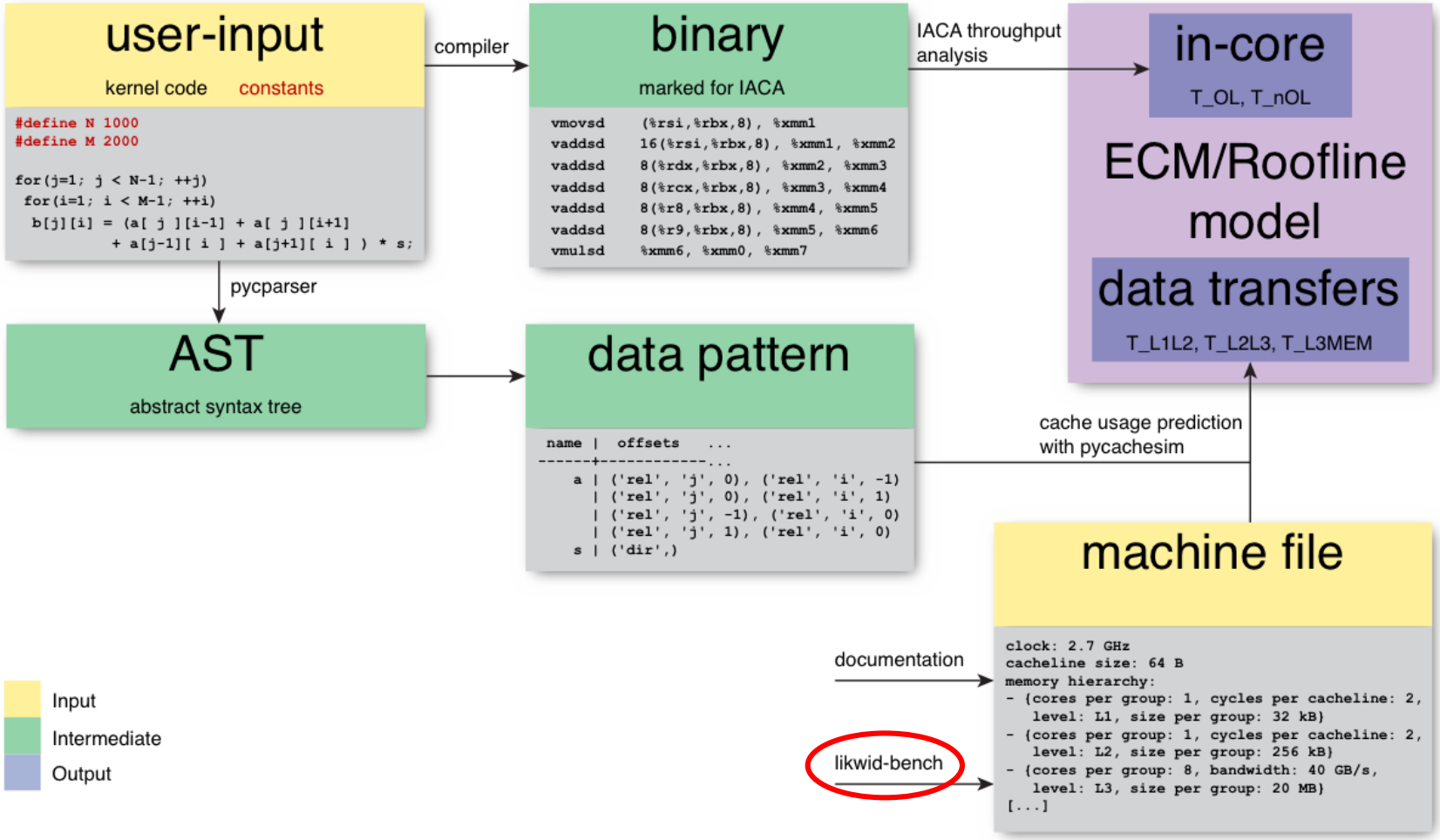


AUTOMATING ANALYTIC MODELS

Kerncraft

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

Kerncraft


user-input
kernel code
constants
#define N 1000
#define M 2000
for(j=1; j < N-1; ++j)
for(i=1; i < M-1; ++i)
b[j][i] = (a[j][i-1] + a[j][i+1]
+ a[j-1][i] + a[j+1][i]) * s;
compiler
binary
marked for IACA
vmovsd (%rsi,%rbx,8), %xmm1
vaddsd 16(%rsi,%rbx,8), %xmm1, %xmm2
vaddsd 8(%rdx,%rbx,8), %xmm2, %xmm3
vaddsd 8(%rcx,%rbx,8), %xmm3, %xmm4
vaddsd 8(%r8,%rbx,8), %xmm4, %xmm5
vaddsd 8(%r9,%rbx,8), %xmm5, %xmm6
vmulsd %xmm6, %xmm0, %xmm7
IACA throughput analysis
in-core
T_OL, T_nOL
ECM/Roofline model
data transfers
T_L1L2, T_L2L3, T_L3MEM
pycparser
AST
abstract syntax tree
data pattern
name | offsets ...
a | ('rel', 'j', 0), ('rel', 'i', -1)
| ('rel', 'j', 0), ('rel', 'i', 1)
| ('rel', 'j', -1), ('rel', 'i', 0)
| ('rel', 'j', 1), ('rel', 'i', 0)
s | ('dir',)
cache usage prediction with pycachesim
machine file
clock: 2.7 GHz
cacheline size: 64 B
memory hierarchy:
- {cores per group: 1, cycles per cacheline: 2, level: L1, size per group: 32 kB}
- {cores per group: 1, cycles per cacheline: 2, level: L2, size per group: 256 kB}
- {cores per group: 8, bandwidth: 40 GB/s, level: L3, size per group: 20 MB}
[...]
documentation
likwid-bench
Input
Intermediate
Output

Machine File

```
model type: Intel Core SandyBridge EP processor
model name: Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
clock: 2.7 GHz
sockets: 2
cores per socket: 8
threads per core: 2
cacheline size: 64 B
micro-architecture: SNB
FLOPs per cycle: {SP: {total: 16, ADD: 8, MUL: 8}
                  DP: {total: 8, ADD: 4, MUL: 4}
overlapping ports: ["0", "0DV", "1", "2", "3", "4", "5"]
non-overlapping ports: ["2D", "3D"]
compiler: icc
compiler flags: [-O3, -xAVX, -fno-alias]

memory hierarchy: […]

benchmarks:
  kernels: […]
  measurements: […]
```

Machine File – Memory Hierarchy

```
[…]
memory hierarchy:
     - level: L1
       cache per group: {'sets': 64, 'ways': 8, 'cl_size': 64, # 32 kB
                         'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True,
                         'load_from': 'L2', 'store_to': 'L2'}
       cores per group: 1
       threads per group: 2
       groups: 16
       cycles per cacheline transfer: 2
     - level: L2
       cache per group: {'sets': 512, 'ways': 8, 'cl_size': 64, # 256 kB
                         'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True,
                         'load_from': 'L3', 'store_to': 'L3'}
       cores per group: 1
       threads per group: 2
       groups: 16
       cycles per cacheline transfer: 2
     - level: L3
       cache per group: {'sets': 20480, 'ways': 16, 'cl_size': 64, # 20 MB
                         'replacement_policy': 'LRU', 'write_allocate': True, 'write_back': True}
       cores per group: 8
       threads per group: 16
       groups: 2
       cycles per cacheline transfer: null
     - level: MEM
       cores per group: 8
       threads per group: 16
[…]
```

Machine File – Benchmark Infos

```
[…]
benchmarks:
  kernels:
    copy:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 8.00 B, streams: 1}
    daxpy:
      FLOPs per iteration: 2
      read streams: {bytes: 16.00 B, streams: 2}
      read+write streams: {bytes: 8.00 B, streams: 1}
      write streams: {bytes: 8.00 B, streams: 1}
    load:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 0.00 B, streams: 0}
    triad:
      FLOPs per iteration: 2
      read streams: {bytes: 24.00 B, streams: 3}
      read+write streams: {bytes: 0.00 B, streams: 0}
      write streams: {bytes: 8.00 B, streams: 1}
    update:
      FLOPs per iteration: 0
      read streams: {bytes: 8.00 B, streams: 1}
      read+write streams: {bytes: 8.00 B, streams: 1}
      write streams: {bytes: 8.00 B, streams: 1}
  measurements: […]
```

Machine File – Benchmark Results

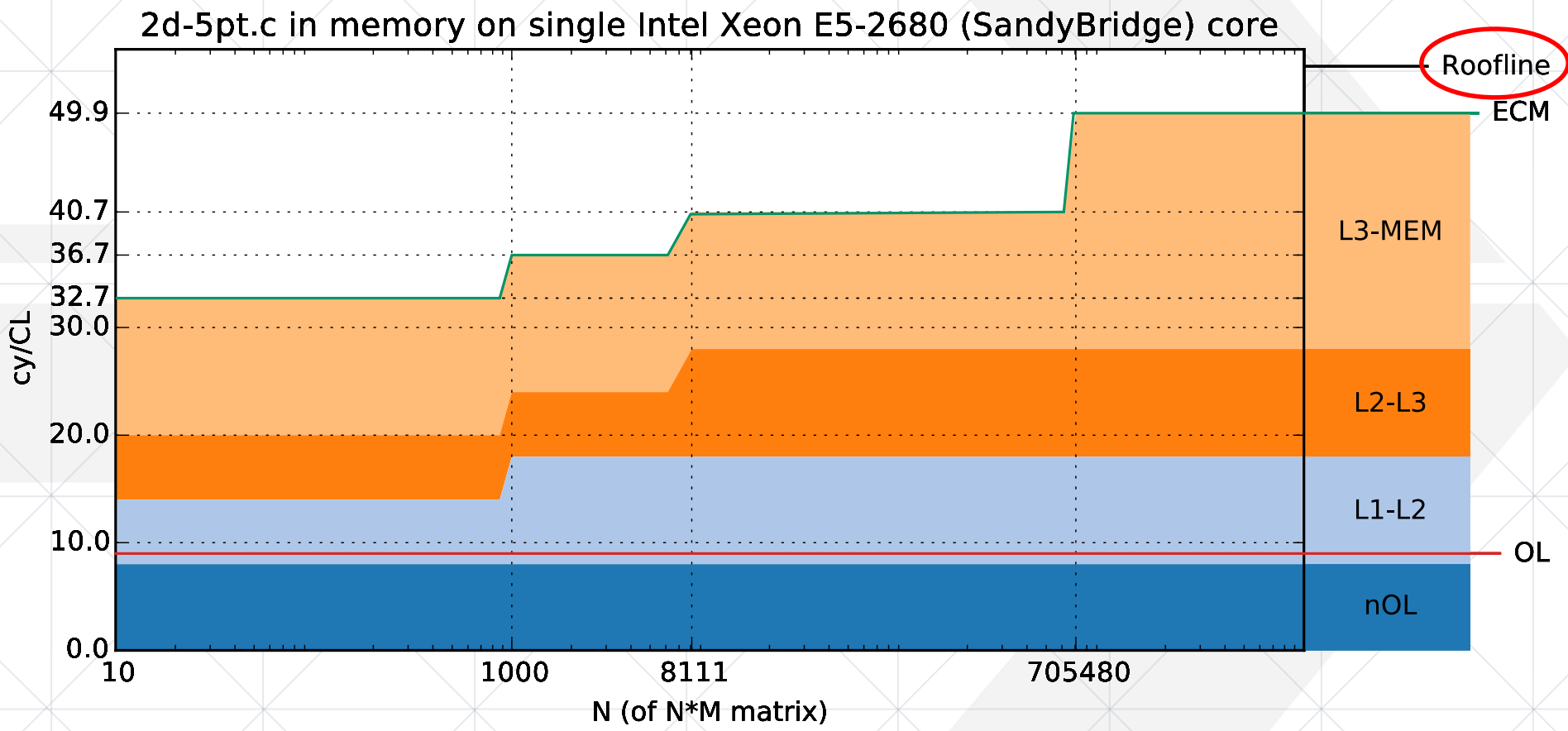
```
[…]
benchmarks:
  kernels: […]
  measurements:
    L1:
      1:
        cores: [1, 2, 3, 4, 5, 6, 7, 8]
        results:
          copy: [81.98 GB/s, 163.75 GB/s, 245.62 GB/s, 327.69 GB/s, 409.41 GB/s,
                 489.83 GB/s, 571.67 GB/s, 653.50 GB/s]
          daxpy: [71.55 GB/s, 143.01 GB/s, 214.86 GB/s, 286.26 GB/s, 355.60 GB/s,
                  426.71 GB/s, 497.45 GB/s, 568.97 GB/s]
[…]
        size per core: [16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB,
                        16.00 kB, 16.00 kB]
        size per thread: [16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB, 16.00 kB,
                          16.00 kB, 16.00 kB]
        threads: [1, 2, 3, 4, 5, 6, 7, 8]
        threads per core: 1
        total size: [16.00 kB, 32.00 kB, 48.00 kB, 64.00 kB, 80.00 kB, 96.00 kB,
                     112.00 kB, 128.00 kB]
[…]
    MEM:
      1:
        cores: [1, 2, 3, 4, 5, 6, 7, 8]
        results:
          copy: [11.60 GB/s, 21.29 GB/s, 25.94 GB/s, 27.28 GB/s, 27.47 GB/s, 27.36
              GB/s, 27.21 GB/s, 27.12 GB/s
[…]
```

Kerncraft – Output

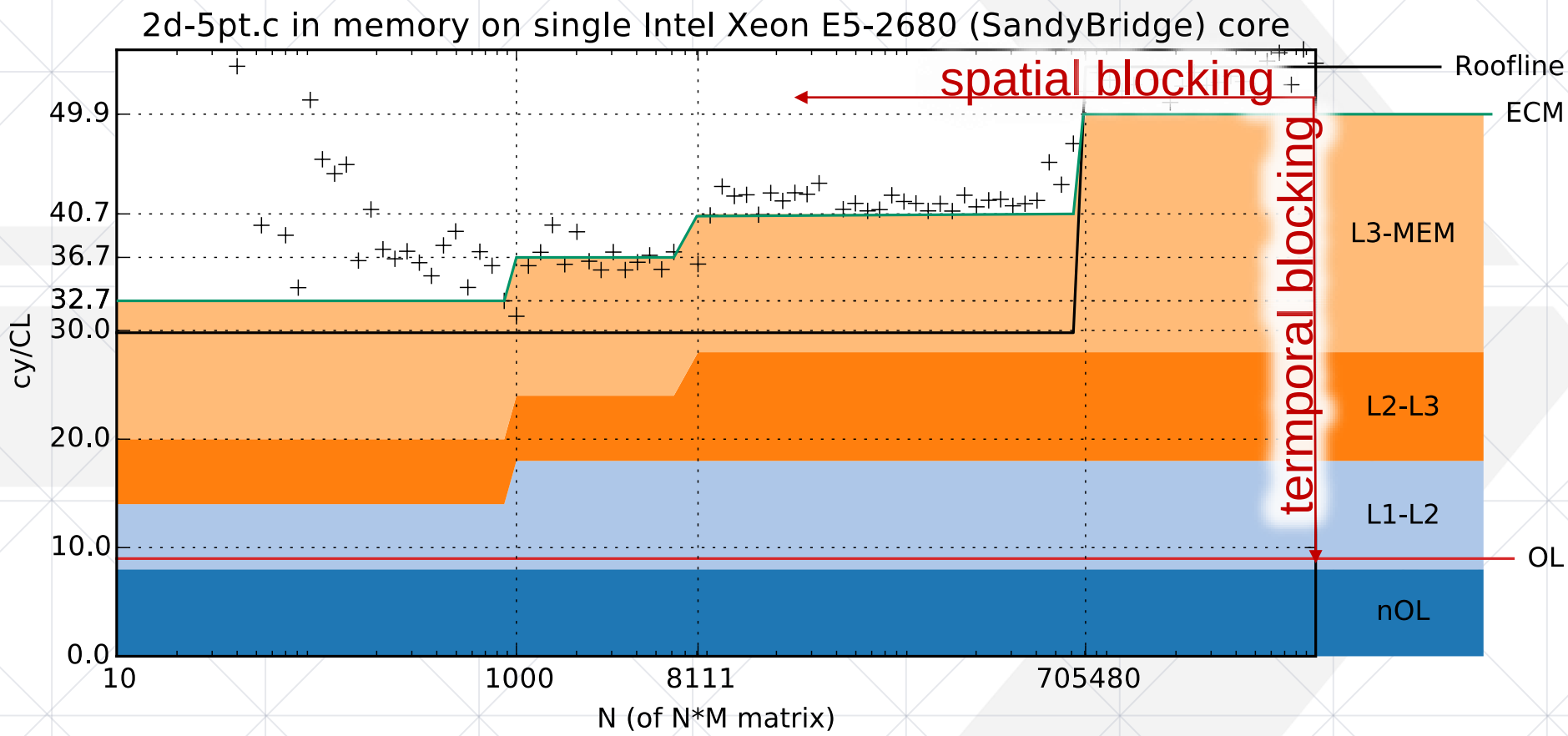
ECM model: $\{ T_{OL} \,||\, T_{nOL} \mid T_{L1\text{-}L2} \mid T_{L2\text{-}L3} \mid T_{L3\text{-}MEM} \}$

```
$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel ECM -D N 5000 -D M 500
================================================================================
kernels/2d-5pt.c
================================================================================
{ 9.0 || 8.0 | 10 | 6 | 12.74 } = 36.74 cy/CL
{ 9.0 \ 18.00 \ 24.00 \ 36.74 } cy/CL
saturating at 3 cores
$
$ kerncraft --machine snb.yaml 2d-5pt.c --pmodel Roofline --unit cy/CL -D N 5000 -D M 500
================================================================================
kernels/2d-5pt.c
================================================================================
Cache or mem bound with 1 core(s)
29.79 cy/CL due to L3-MEM transfer bottleneck (bw from copy benchmark)
Arithmetic Intensity: 0.17 FLOP/b
$
```

Kerncraft – Results



Kerncraft – Results



So what should I take home from this?

- A failure of an analytic performance model is not the end of the world
  - It is an opportunity to learn more
- Analytic modeling takes longer than running the code. Why should I bother?
  - The model is not just a number; it is insight
  - Newton could have measured the time of 1000 apples falling to the floor, but that would not have led him to the law of gravitation
- Why not use cycle-accurate simulators?
  - Good luck with laying your hands on one of those
  - Lowest-order effects may be hidden under heaps of data

ERLANGEN REGIONAL COMPUTING CENTER

SPONSORED BY THE
Federal Ministry of Education and Research

MetaXa

KONWIHR III
Bavarian Network for HPC

Thank You.

Julian Hammer
Holger Stengel
Jan Eitzinger
Gerhard Wellein

Further pointers

- (Semi-) Automatic modeling of streaming kernels with Kerncraft
  - https://github.com/RRZE-HPC/kerncraft
- LIKWID toolkit for HPM measurements (and much more)
  - https://github.com/RRZE-HPC/likwid
- Layer condition and block size calculator for stencil codes
  - https://rrze-hpc.github.io/layer-condition/
- Girih test harness for temporally blocked stencil algorithms
  - https://github.com/ecrc/girih

Further references

- J. Treibig and G. Hager: *Introducing a Performance Model for Bandwidth-Limited Loop Kernels*. Proceedings of the Workshop “Memory issues on Multi- and Manycore Platforms” at PPAM 2009, the 8th International Conference on Parallel Processing and Applied Mathematics, Wroclaw, Poland, September 13-16, 2009. Lecture Notes in Computer Science Volume 6067, 2010, pp 615-624. DOI: 10.1007/978-3-642-14390-8_64 (2010).
- J. Treibig, G. Wellein and G. Hager: *Efficient multicore-aware parallelization strategies for iterative stencil computations*. Journal of Computational Science 2, 130-137 (2011). DOI: 10.1016/j.jocs.2011.01.010
- G. Hager, J. Treibig, J. Habich, and G. Wellein: *Exploring performance and power properties of modern multicore chips via simple machine models*. Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.3180 (2013).
- M. Wittmann, G. Hager, T. Zeiser, J. Treibig, and G. Wellein: *Chip-level and multi-node analysis of energy-optimized lattice-Boltzmann CFD simulations.* Concurrency and Computation: Practice and Experience 28(7), 2295-2315 (2015). DOI: 10.1002/cpe.3489
- T. M. Malas, G. Hager, H. Ltaief, and D. E. Keyes: *Multi-dimensional intra-tile parallelization for memory-starved stencil computations*. Accepted for publication in ACM Transactions on Parallel Computing. Preprint: arXiv:1510.04995

Further references

- M. Wittmann, G. Hager, J. Treibig and G. Wellein: *Leveraging shared caches for parallel temporal blocking of stencil codes on multicore processors and clusters.* Parallel Processing Letters **20** (4), 359-376 (2010). DOI: 10.1142/S0129626410000296
- J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein: *Pushing the limits for medical image reconstruction on recent standard multicore processors.* International Journal of High Performance Computing Applications **27**(2), 162-177 (2013). DOI: 10.1177/1094342012442424
- T. M. Malas, G. Hager, H. Ltaief, H. Stengel, G. Wellein, and D. E. Keyes: *Multicore-optimized wavefront diamond blocking for optimizing stencil updates*. SIAM Journal on Scientific Computing **37**(4), C439-C464 (2015). DOI: 10.1137/140991133
- J. Hammer, G. Hager, J. Eitzinger, and G. Wellein: *Automatic Loop Kernel Analysis and Performance Modeling With Kerncraft*. Proc. PMBS15, in conjunction with SC15, November 16, 2015, Austin, TX. DOI: 10.1145/2832087.2832092
- J. Hammer, J. Eitzinger, G. Hager, and G. Wellein: *Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels*. Proc. IPTW 2016, the 10th International Parallel Tools Workshop, October 4-5, 2016, Stuttgart, Germany. DOI: 10.1007/978-3-319-56702-0_1, Preprint: arXiv:1702.04653